```
import cv2
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import img_to_array
import numpy as np
```

در این قسمت از کد، کتابخانه‌ها و ماژول‌های مورد نیاز برای اجرای برنامه به وسیله `import` به برنامه اضافه شده‌اند. دیگر چیزهایی که این بخش انجام می‌دهند به شرح زیر است:

1. `import cv2`: این خط کتابخانه OpenCV را به برنامه اضافه می‌کند. OpenCV یک کتابخانه معروف در زمینه بینایی ماشین و پردازش تصویر است و از آن برای انجام وظایفی مانند تشخیص چهره، تشخیص اجسام، ویدئوپردازش و غیره استفاده می‌شود.

2. `from keras.models import load_model`: این خط از کتابخانه Keras یک ماژول به نام `load_model` را وارد می‌کند. Keras یک چارچوب عالی برای ساخت، آموزش و اجرای شبکه‌های عصبی عمیق است. `load_model` نیز یک تابع است که برای بارگذاری مدل‌های آموزش دیده شده از پیش در Keras استفاده می‌شود.

Keras یک کتابخانه متن‌باز (open-source) برای ساخت، آموزش، و اجرای شبکه‌های عصبی در پایتون است. این کتابخانه ابتدا توسط François Chollet توسعه یافته است و اکنون تحت پروژه TensorFlow قرار دارد. Keras از زبان‌های برنامه‌نویسی مختلف مانند TensorFlow، Theano و Microsoft Cognitive Toolkit پشتیبانی می‌کند.

ویژگی‌های مهم Keras عبارتند از:

1. سهولت استفاده: Keras با تمرکز بر روی سهولت استفاده از شبکه‌های عصبی، این امکان را به برنامه‌نویسان می‌دهد تا به راحتی مدل‌های عمیق بسازند و با آن‌ها کار کنند.

2. قابلیت چند پشت‌انداز: Keras به عنوان یک واسط مستقل عمل می‌کند و می‌تواند روی پشت‌اندازهای مختلفی مانند TensorFlow یا Theano اجرا شود.

پشت‌انداز یا Backend در مفهوم کتابخانه‌های یادگیری عمیق (Deep Learning) به سیستمی اشاره دارد که عملیات محاسباتی (operations) شبکه عصبی را پیاده‌سازی می‌کند. در مواقعی که ما از یک کتابخانه یادگیری عمیق مثل TensorFlow یا Theano یا MXNet استفاده می‌کنیم، این کتابخانه‌ها به عنوان پشت‌انداز (Backend) برای اجرای محاسبات مورد نیاز در شبکه‌های عصبی عمیق عمل می‌کنند.

به عبارت دیگر، پشت‌انداز مسئولیت اجرای محاسبات مربوط به شبکه عصبی را برعهده دارد. این محاسبات شامل عملیات‌های مانند جمع و تفریق ماتریس‌ها، ضرب ماتریسی، توابع فعال‌سازی، به‌روزرسانی وزن‌ها، بهینه‌سازی و ... است.

برخی از پشت‌اندازهای معروف مورد استفاده در کتابخانه‌های یادگیری عمیق عبارتند از:

1. TensorFlow Backen: این پشت‌انداز توسط TensorFlow استفاده می‌شود.

2. heano Backen: پشت‌انداز Theano که توسط یک گروه تحقیقاتی در دانشگاه مونترال ساخته شده است.

3. MXNet Backen: این پشت‌انداز توسط Apache MXNet استفاده می‌شود.

4. CNTK Backend: مایکروسافت Cognitive Toolkit (CNTK) نیز یک پشت‌انداز محبوب است.

معمولاً کاربران با انتخاب یک پشت‌انداز خاص، مثلاً TensorFlow، به‌راحتی می‌توانند از ابزارها و ویژگی‌های ارائه شده توسط آن کتابخانه استفاده کنند. اما این امکان همچنین برخی از موارد را به آنها می‌دهد که از پشت‌اندازهای مختلفی استفاده کنند و با مزایا و معایب هرکدام آشنا شوند.

3. **پشتیبانی از توزیع موازی**: Keras بر روی توزیع‌های موازی برای آموزش مدل‌های بزرگ و پیچیده نیز پشتیبانی می‌کند.

توزیع موازی (Parallel Computing) فرآیند انجام محاسبات بزرگ را با استفاده از چندین دستگاه (یا هسته) همزمان انجام می‌دهد. هدف از توزیع موازی بهبود عملکرد و سرعت اجرای الگوریتم‌ها و محاسبات است. در سیستم‌های توزیع موازی، وظایف مختلف به چندین دستگاه مستقل اختصاص می‌یابد و هر یک از این دستگاه‌ها به‌صورت همزمان بر روی بخش‌های مختلف محاسبات کار می‌کنند.

توزیع موازی می‌تواند در سطح یک دستگاه (مانند چندین هسته در یک پردازنده) یا در سطح شبکه‌های محلی (LAN) و یا حتی در سطح شبکه‌های گسترده (WAN) صورت بگیرد. برخی از مفاهیم مهم توزیع موازی عبارتند از:

1. کارمند-رئیس (Master-Worker): یک کامپیوتر یا نهاد مسئولیت تقسیم کار را برعهده دارد (کارمند) و دیگر کامپیوترها (رئیسان) وظیفه‌های اختصاص یافته به آن‌ها را اجرا می‌کنند.

2. تقسیم و حکومت (Divide and Conquer): مسئله بزرگ‌تر را به زیرمسائل کوچکتر تقسیم می‌کند و هر کدام از این زیرمسائل به‌طور موازی حل می‌شوند.

3.ردازش‌های موازی (Parallel Processing): بیش از یک عملیات همزمان روی داده‌ها انجام می‌شود.

4.بادل داده موازی (Parallel Data Processing): داده‌ها به‌صورت همزمان توسط چندین واحد پردازشی پردازش می‌شوند.

5.وزیع بار (Load Balancing): تساوی بار کاری بین دستگاه‌ها یا هسته‌ها به‌صورت مناسب.

6. مدیریت تراکنش‌های موازی (Parallel Transaction Management): مدیریت تراکنش‌ها در محیط‌های توزیع موازی.

استفاده از توزیع موازی معمولاً در مواردی که نیاز به پردازش بزرگ و محاسبات گسترده داریم، مثل یادگیری عمیق (Deep Learning)، شبیه‌سازی‌ها، مدل‌های ریاضی پیچیده، و تحلیل داده‌های حجیم (Big Data) مفید است. این رویکرد بهبود قابلیت مقیاس‌پذیری و عملکرد در برنامه‌ها و سامانه‌ها را ارتقا می‌دهد.

4. سازگاری با TensorFlow 2.x: با انتقال Keras به TensorFlow 2.x به عنوان پشتیبان توسعه‌ای که توسط TensorFlow ارائه می‌شود، Keras به‌طور رسمی به عنوان یک بخش اصلی از TensorFlow در نسخه‌های جدیدتر این کتابخانه یادگیری عمیق شده است.

از زمان ادغام Keras به TensorFlow، بسیاری از برنامه‌نویسان و پژوهشگران از این ترکیب برای توسعه و آموزش مدل‌های عصبی بهره‌مند شده‌اند.

3. `from time import sleep`: این خط ماژول `sleep` از کتابخانه `time` را وارد می‌کند. این ماژول به برنامه این امکان را می‌دهد که به مدت زمان مشخصی تاخیر داشته باشد. در اینجا ممکن است برای مدیریت زمان در حلقه برنامه یا ایجاد تاخیرهای مد نظر استفاده شده باشد.

4. `from keras.preprocessing.image import img_to_array`: این خط ماژول `img_to_array` از کتابخانه Keras را وارد می‌کند. این ماژول تابعی به نام `img_to_array` را فراهم می‌کند که تصویر را به یک آرایه NumPy تبدیل می‌کند. این عمل برای تهیه داده‌های ورودی مناسب برای مدل استفاده می‌شود.

5. `import numpy as np`: این خط کتابخانه NumPy را وارد برنامه می‌کند و آن را با نام مخفف `np` فراخوانی می‌کند. NumPy یک کتابخانه محاسبات عددی در پایتون است و اغلب برای انجام عملیات ماتریسی و آرایه‌های عددی در پروژه‌های علم داده و یادگیری ماشین استفاده می‌شود.

```
# Load the pre-trained emotion detection model
classifier = load_model(r'model.h5')

# Load the Haar Cascade face classifier
face_classifier =
cv2.CascadeClassifier(r'haarcascade_frontalface_default.xml')

# Define emotion labels
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad',
'Surprise']

# Set up webcam capture
```

```
cap = cv2.VideoCapture(0)

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480))

# Initialize color and emotion label outside the loop
color = (0, 255, 255)
```

1. `classifier = load_model(r'model.h5')`: در این خط، مدل تشخیص احساسات از پیش آموزش دیده که در فایل `model.h5` ذخیره شده است را بارگذاری می‌کند. تابع `load_model` از کتابخانه Keras برای بارگذاری یک مدل از فایل استفاده می‌شود. این مدل با استفاده از شبکه عصبی عمیق آموزش دیده شده برای تشخیص احساسات در تصاویر کار می‌کند.

2. `face_classifier = cv2.CascadeClassifier(r'haarcascade_frontalface_default.xml')`: این خط از کتابخانه OpenCV، یک کلاسیفایر چهره به نام Haar Cascade را بارگذاری می‌کند. این کلاسیفایر از یک مدل آموزش دیده شده برای تشخیص چهره‌ها با استفاده از الگوریتم Haar Cascade استفاده می‌کند. فایل `haarcascade_frontalface_default.xml` حاوی پارامترها و اطلاعات مورد نیاز برای تشخیص چهره است.

3. `emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']`: این خط یک لیست از برچسب‌های مرتبط با هر احساس را تعریف می‌کند. این برچسب‌ها برای نمایش نتایج تشخیص احساسات روی تصویرها استفاده می‌شوند.

4. `cap = cv2.VideoCapture(0)`: این خط یک شیء VideoCapture ایجاد می‌کند که از دوربین وب استفاده می‌کند. عدد 0 به معنای انتخاب دوربین پیش‌فرض است.

5. `fourcc = cv2.VideoWriter_fourcc(*'XVID')`: این خط یک چهارک کد (FourCC) برای نوشتن ویدئو مشخص می‌کند. در اینجا از کدک XVID برای فرمت ویدئو استفاده شده است.

6. `out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480))`: این خط یک شیء VideoWriter ایجاد می‌کند که برای نوشتن ویدئو خروجی به نام `output.avi` با استفاده از چهارک کد XVID و با فریم ریت 20.0 فریم در ثانیه و ابعاد 480*640 استفاده می‌شود.

7. `color = (0, 255, 255)`: این خط یک تاپل شامل مقادیر RGB برای رنگ ابتدایی تعیین می‌کند. در اینجا (0, 255, 255) متناظر با رنگ زرد است. این رنگ برای رسم مستطیل و برچسب احساسات بر روی چهره‌ها استفاده می‌شود.

این متغیرها به ترتیب برای بارگذاری مدل تشخیص احساسات، تشخیص چهره، تعیین برچسب‌های احساسات، گرفتن تصاویر از دوربین، تنظیم فرمت ویدئو و تعیین رنگ برای نمایش نتایج استفاده می‌شوند.

```
while True:
    _, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the frame
    faces = face_classifier.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))

    for i, (x, y, w, h) in enumerate(faces):
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
        roi_gray = gray[y:y + h, x:x + w]
        roi_gray = cv2.resize(roi_gray, (48, 48),
interpolation=cv2.INTER_AREA)

        # If the region of interest is not empty, perform emotion prediction
        if np.sum([roi_gray]) != 0:
            roi = roi_gray.astype('float') / 255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi, axis=0)

            prediction = classifier.predict(roi)[0]
            label = emotion_labels[prediction.argmax()]

            # Update the label position to consider the face position
            label_position = (x, y - 10)

            # Use different colors for different faces
            color = (np.random.randint(0, 255), np.random.randint(0, 255),
np.random.randint(0, 255))

            cv2.putText(frame, label, label_position,
cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
        else:
            cv2.putText(frame, 'No Faces', (30, 80),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
```

1. `cap.read()`: این دستور از دوربین وب تصویر گرفته و آن را در متغیر `frame` ذخیره می‌کند. خروجی `_` نشان‌دهنده مقدار بازگشتی دوم تابع `read` است که در اینجا صرفاً نادیده گرفته شده است.

2. `cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)`: این خط تصویر رنگی `frame` را به تصویر خاکستری تبدیل می‌کند. این گام معمولاً برای ساده‌سازی پردازش تصویر و کاهش تعداد کانال‌های رنگی مورد استفاده قرار می‌گیرد.

3. `face_classifier.detectMultiScale(...)`: این خط از کلاسیفایر چهره Haar Cascade استفاده می‌کند تا چهره‌ها را در تصویر شناسایی کند. نتایج در متغیر `faces` ذخیره می‌شوند.

4. `for i, (x, y, w, h) in enumerate(faces)`: یک حلقه `for` بر روی چهره‌های تشخیص داده شده اجرا می‌شود. این حلقه همچنین از `enumerate` برای دریافت شماره‌ی ترتیبی چهره‌ها نیز استفاده می‌کند.

5. `cv2.rectangle(...)`: این خط یک مستطیل دور چهره را بر روی تصویر اصلی (`frame`) رسم می‌کند.

6. `roi_gray = gray[y:y + h, x:x + w]`: این خط ناحیه‌ی منافذ (ROI) را که تنها شامل چهره است، از تصویر خاکستری `gray` استخراج می‌کند.

7. `cv2.resize(...)`: این خط ناحیه منافذ را به ابعاد مشخصی (در اینجا48*48 پیکسل) تغییر اندازه می‌دهد. این ابعاد به عنوان ورودی به مدل تشخیص احساسات مورد استفاده قرار می‌گیرد.

8. `if np.sum([roi_gray]) != 0`: این شرط بررسی می‌کند که آیا ناحیه منافذ خالی است یا خیر. اگر خالی نباشد، به تشخیص احساسات می‌پردازد.

9. `prediction = classifier.predict(roi)[0]`: این خط احساسات

موجود در ناحیه منافذ را با استفاده از مدل تشخیص احساسات پیش‌بینی می‌کند.

10. `label = emotion_labels[prediction.argmax()]`: این خط برچسب مرتبط با احساسات پیش‌بینی شده را از میان برچسب‌های تعریف شده (`emotion_labels`) انتخاب می‌کند.

11. `label_position = (x, y - 10)`: موقعیت برچسب بر اساس موقعیت چهره تنظیم می‌شود.

12. `color = (np.random.randint(0, 255), np.random.randint(0, 255), np.random.randint(0, 255))`: رنگ جدیدی برای نمایش نتایج در هر چهره انتخاب می‌شود.

13. `cv2.putText(...)`: این خط برچسب احساسات را با رنگ و موقعیت محاسبه شده روی تصویر اصلی (`frame`) قرار می‌دهد.

14. `cv2.putText(frame, 'No Faces', (30, 80), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)`: این خط یک پیام "هیچ چهره‌ای یافت نشد" را نمایش می‌دهد اگر هیچ چهره‌ای در تصویر تشخیص داده نشود.

این عملیات به صورت مکرر در حلقه انجام می‌شود تا همیشه تصاویر جدید از دوربین گرفته شده و پردازش احساسات روی آن‌ها ادامه یابد.

```
    # Write the frame to the output video
    out.write(frame)

    # Show the frame
    cv2.imshow('Emotion Detector', frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the webcam, VideoWriter, and close all windows
cap.release()
out.release()
cv2.destroyAllWindows()
```

1. `out.write(frame)`: این خط فریم پردازش شده (با برچسب احساسات) را به فایل ویدئو خروجی (`output.avi`) می‌نویسد. این فرآیند برای هر فریم در حلقه تکرار می‌شود، که نهایتاً یک ویدئو با نتایج پردازش احساسات ایجاد می‌شود.

2. `cv2.imshow('Emotion Detector', frame)`: این خط فریم پردازش شده را در یک پنجره نمایش می‌دهد. پنجره با نام 'Emotion Detector' ایجاد می‌شود.

3. `if cv2.waitKey(1) & 0xFF == ord('q')`: این شرط بررسی می‌کند که آیا کاربر کلید 'q' را فشرده است یا خیر. اگر کلید 'q' فشرده شده باشد، حلقه برنامه متوقف شده و برنامه به پایان می‌رسد.

4. `cap.release()`: این خط دوربین وب را آزاد می‌کند، به این ترتیب دیگر دسترسی به دوربین نخواهیم داشت.

5. `out.release()`: این خط ویدئونویس را آزاد می‌کند. پس از اجرای این دستور، ویدئو خروجی ذخیره می‌شود.

6. `cv2.destroyAllWindows()`: این خط تمام پنجره‌هایی که با استفاده از OpenCV ایجاد شده‌اند را بسته و منابع را آزاد می‌کند.

به این ترتیب، برنامه پایان می‌یابد و تمام منابع، چه دوربین وب و چه فایل ویدئو، آزاد می‌شوند.

بله، حتما. در ادامه توضیحات بخط به خط برای این بخش از کد آورده شده‌اند:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
```

1. **matplotlib.pyplot**: این کتابخانه برای تولید نمودارها و تصاویر استفاده می‌شود.

2. **numpy (np)**: این کتابخانه برای انجام عملیات عددی و ماتریسی، به خصوص در پردازش تصاویر و داده‌های عددی، استفاده می‌شود.

3. **pandas (pd)**: این کتابخانه برای کار با داده‌ها به صورت داده فریم (data frame) مانند جداول استفاده می‌شود.

4. **seaborn**: این کتابخانه برای تولید نمودارهای زیبا و افزودن انیمیشن به آن‌ها به کار می‌رود.

5. **os**: این کتابخانه برای ارتباط با سیستم عامل (مثل خواندن و نوشتن فایل‌ها و پوشه‌ها) استفاده می‌شود.

```
from keras.preprocessing.image import load_img, img_to_array
```

6. **keras.preprocessing.image.load_img**: این تابع برای بارگذاری تصویر از یک فایل به عنوان یک شیء پیکربندی (PIL Image) به کار می‌رود.

7. **keras.preprocessing.image.img_to_array**: این تابع برای تبدیل تصویر از شیء پیکربندی (PIL Image) به یک آرایه نامپای (NumPy array) استفاده می‌شود.

```
from keras.preprocessing.image import ImageDataGenerator
```

8. **keras.preprocessing.image.ImageDataGenerator**: این کلاس برای تولید داده‌های جدید (آگمانتیشن) از داده‌های موجود برای آموزش مدل‌های شبکه عصبی عمیق به کار می‌رود.

```
from keras.layers import Dense, Input, Dropout, GlobalAveragePooling2D, Flatten, Conv2D, BatchNormalization, Activation, MaxPooling2D
```

9. **keras.layers**: این ماژول محاسبات لایه‌ای (layers) در معماری مدل‌های شبکه عصبی را فراهم می‌کند. مثلاً Dense برای افزودن یک لایه متصل کاملاً و Conv2D برای افزودن لایه کانولوشنال به کار می‌رود.

10. **keras.models**: این ماژول برای تعریف و استفاده از مدل‌های شبکه عصبی به کار می‌رود. برای مثال، Sequential برای ایجاد یک مدل به صورت ترتیبی.

11. **keras.optimizers**: این ماژول برای تعیین و تنظیم الگوریتم بهینه‌سازی برای آموزش مدل‌ها استفاده می‌شود. در اینجا از Adam، SGD، و RMSprop به عنوان بهینه‌سازها استفاده شده‌اند.

```
model = Sequential()
```

12. **keras.models.Sequential**: این کلاس برای ایجاد یک مدل شبکه عصبی به صورت ترتیبی استفاده می‌شود.

```
picture_size = 48
folder_path = "../input/face-expression-recognition-dataset/images/"
expression = 'disgust'
```

13. **picture_size**: اندازه تصاویر ورودی به مدل (48x48) را مشخص می‌کند.

14. **folder_path**: مسیر داده‌های تصاویر را مشخص می‌کند.

15. **expression**: احساس مورد نظر (در اینجا 'disgust' یا انزجار) برای نمایش تصاویر آموزشی انتخاب شده است.

```
plt.figure(figsize= (12,12))
for i in range(1, 10, 1):
    plt.subplot(3,3,i)
    img = load_img(folder_path+"train/"+expression+"/"+os.listdir(folder_path + "train/" + expression)[i], target_size=(picture_size, picture_size))
    plt.imshow(img)
plt.show()
```

16. **matplotlib.pyplot.figure**: ایجاد یک شیء شکل (figure) به ابعاد خاص.

17. **matplotlib.pyplot.subplot**: ایجاد یک زیرنمودار در داخل شکل.

18. **keras.preprocessing.image.load_img**: بارگذاری تصویر به عنوان یک شیء پیکربندی (PIL Image).

19. **matplotlib.pyplot.imshow**: نمایش تصویر در زیرنمودار.

20. **matplotlib.pyplot.show**: نمایش شکل (figure) به صورت نهایی.

```
batch_size = 128

datagen_train = ImageDataGenerator()
datagen_val = ImageDataGenerator()
```

21. **batch_size**: تعداد تصاویر در هر دسته برای آموزش مدل.

22. **keras.preprocessing.image.ImageDataGenerator**: ایجاد یک نمونه از این کلاس برای تولید داده‌های جدید (آگمانتیشن) در هنگام آموزش.

```
,"train_set = datagen_train.flow_from_directory(folder_path+"train
,target_size = (picture_size,picture_size)
,"color_mode = "grayscale
,batch_size=batch_size
,'class_mode='categorical
(shuffle=True
```

23. **train_set**: یک ژنراتور داده برای داده‌های آموزش ایجاد می‌شود.

24. **datagen_train.flow_from_directory**: این تابع برای تولید داده‌های آموزش از داده‌های موجود در یک دایرکتوری استفاده می‌شود.

```
,"test_set = datagen_val.flow_from_directory(folder_path+"validation
,target_size = (picture_size,picture_size)
,"color_mode = "grayscale
,batch_size=batch_size
,'class_mode='categorical
(shuffle=False
```

25. **test_set**: یک ژنراتور داده برای داده‌های ارزیابی ایجاد می‌شود.

26. **datagen_val.flow_from_directory**: این تابع برای تولید داده‌های ارزیابی از داده‌های موجود در یک دایرکتوری استفاده می‌شود.

```python
from keras.optimizers import Adam,SGD,RMSprop
```

```

27. **keras.optimizers.Adam, SGD, RMSprop**: الگوریتم‌های بهینه‌سازی مختلف برای آموزش مدل. در اینجا، از Adam به عنوان الگوریتم بهینه‌سازی استفاده شده است.

```
no_of_classes = 7

()model = Sequential

st CNN layer1 #
model.add(Conv2D(64, (3,3), padding='same', input_shape=(48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

nd CNN layer2 #
model.add(Conv2D(128, (5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

rd CNN layer3 #
model.add(Conv2D(512, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
```
```

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

th CNN layer4 #
model.add(Conv2D(512, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())

Fully connected 1st layer #
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

Fully connected 2nd layer #
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(no_of_classes, activation='softmax'))
```

28. **no_of_classes**: تعداد کلاس‌ها (در اینجا 7 کلاس برای احساسات مختلف).

29. **Sequential**: ایجاد یک مدل شبکه عصبی به صورت ترتیبی.

30. **Conv2D**: افزودن یک لایه کانولوشنال به مدل.

31. **BatchNormalization**: افزودن یک لایه نرمال‌سازی دسته‌ای.

32. **Activation('relu')**: افزودن تابع فعال‌سازی ReLU به لایه.

33. **MaxPooling2D**: افزودن یک لایه ادغام (Pooling) به مدل.

34. **Dropout(0.25)**: افزودن یک لایه Dropout برای کاهش اتصالات غیرضروری و جلوگیری از بیش‌برازش.

35. **Flatten()**: افزودن یک لایه Flatten برای تبدیل ویژگی‌های چند بعدی به یک بردار.

36. **Dense**: افزودن یک لایه کاملاً متصل به مدل.

37. **Activation('softmax')**: تابع فعال‌سازی softmax برای تبدیل خروجی به احتمالات.

```
opt = Adam(lr=0.0001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
()model.summary
```

38. **Adam(lr=0.0001)**: تعیین نرخ یادگیری برای الگوریتم بهینه‌سازی Adam.

39. **model.compile**: کامپایل مدل با تنظیمات بهینه‌ساز، تابع هزینه (در اینجا categorical_crossentropy) و معیار (در اینجا accuracy).

40. **model.summary()**: نمایش خلاصه‌ای از معماری مدل.

```
from keras.optimizers import RMSprop, SGD, Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

checkpoint = ModelCheckpoint("./model.h5", monitor='val_acc', verbose=1, save_best_only=True, mode='max')

,'early_stopping = EarlyStopping(monitor='val_loss
                                 ,min_delta=0
                                 ,patience=3
                                 ,verbose=1
                                 restore_best_weights=True
                                 (

,'reduce_learningrate = ReduceLROnPlateau(monitor='val_loss
                                 ,factor=0.2
                                 ,patience=3
                                 ,verbose=1
                                 (min_delta=0.0001

callbacks_list = [early_stopping, checkpoint, reduce_learningrate]
```

41. **ModelCheckpoint**: ذخیره‌سازی بهترین مدل بر اساس عملکرد در حین آموزش.

42. **EarlyStopping**: قطع آموزش زودهنگام در صورت عدم بهبود عملکرد.

43. **ReduceLROnPlateau**: کاهش نرخ یادگیری در صورت عدم بهبود عملکرد.

44. **callbacks_list**: یک لیست از کلیه callback ها برای استفاده در آموزش مدل.

```
epochs = 48

,'model.compile(loss='categorical_crossentropy
                ,optimizer=Adam(lr=0.001)
                (metrics=['accuracy']

,history = model.fit_generator(generator=train_set
        ,steps_per_epoch=train_set.n//train_set.batch_size
                ,epochs=epochs
                ,validation_data=test_set
        ,validation_steps=test_set.n//test_set.batch_size
                (callbacks=callbacks_list
```

45. **epochs**: تعداد دوره‌های آموزش.

46. **model.fit_generator**: آموزش مدل با ژنراتور داده‌های آموزش و ارزیابی بر روی داده‌های ارزیابی.

این بخش‌ها از کد برای تعریف مدل، تنظیمات آموزش، و استفاده از callback ها برای کنترل آموزش مدل استفاده شده‌اند.